Hybrid Logical Clocks for Database Forensics

Filling the Gap between Chain of Custody and Database Auditing

Denys A. Flores*† and Arshad Jhumka*
*University of Warwick. Department of Computer Science. Coventry, United Kingdom
{d.flores-armas, h.a.jhumka}@warwick.ac.uk
†Escuela Politécnica Nacional. Departmento de Informática y Ciencias de la Computación (DICC). Quito, Ecuador
denys.flores@epn.edu.ec

***Abstract*—Database audit records are important for investigating suspicious actions against transactional databases. Their admissibility as digital evidence depends on satisfying Chain of Custody ($\mathcal{C}o\mathcal{C}$) properties during their generation, collection and preservation in order to prevent their modification, guarantee action accountability, and allow third-party verification. However, their production has relied on auditing capabilities provided by commercial database systems which may not be effective if malicious users (or insiders) misuse their privileges to disable audit controls, and compromise their admissibility. Hence, in this paper, we propose a forensically-aware distributed database architecture that implements $\mathcal{C}o\mathcal{C}$ properties as functional requirements to produce admissible audit records. The novelty of our proposal is the use of hybrid logical clocks, which compared with a previous centralised vector-clock architecture, has evident advantages as it (i) allows for more accurate provenance and causality tracking of insider actions, (ii) is more scalable in terms of system size, and (iii) although latency is higher (as expected in distributed environments), 70 per cent of user transactions are executed within acceptable latency intervals.**

***Index Terms*—database audit, database forensics, database security, hybrid logical clocks, chain of custody, provenance, causality, role segregation.**

I. Introduction

In 1989, research about the efficient design of computing architectures for processing secure database transactions was already proposed as part of OS kernel extensions [1]. Currently, user applications and software development needs, have evolved these controls from obscure kernel primitives to user-accessible audit functionality in commercial transactional databases [2][3] in order to secure distributed transaction processing. Conversely, when security of such transactions is compromised, database forensics has emerged as a reactive approach to investigate (obtain, analyse and present in court) digital evidence about the attribution of actions to malicious users [4], which may not be fully compatible when dealing with complex database structures [5]. Instead, a proactive approach for the generation, collection and preservation of database audit records seems more suitable for capturing the occurrence of $\mathcal{DML}$ operations (events) during the normal operation of a forensically-aware database architecture [6]. Particularly, for making these audit records forensically admissible as digital evidence, this approach enforces Chain of Custody ($\mathcal{C}o\mathcal{C}$) properties [8] as (1) role segregation to prevent their modification, (2) capturing provenance of every event performed on transactional data, (3) event ordering in a timeline for explaining their causality and allowing third-party verification, and (4) ensuring these properties remain active during the architecture's normal operation. However, if malicious users (insiders) misuse their privileges [7] in order to disable any audit functionality and compromise a transactional database, $\mathcal{C}o\mathcal{C}$ properties are affected, making any audit records inadmissible as evidence. Therefore, in this paper, we propose, deploy and evaluate a forensically-aware distributed database architecture designed to handle suspicious $\mathcal{DML}$ operations when insiders misuse their credentials to alter a database state without being detected. This novel architecture is based on hybrid logical clocks ($\mathbb{HLC}$), first proposed in [9], but implementing $\mathcal{C}o\mathcal{C}$ properties at its core to support the generation, collection and preservation of audit records, having a transactional and a forensic database residing on separate servers. As opposed to a previous centralised vector-clock ($\mathbb{VC}$) based architecture [6], an experimental deployment of our proposal shows that (i) provenance and causality tracking is indeed more accurate, preventing inconsistent observations since concurrent and sequential events are timelined using synchronised physical clocks (ii) timestamps are node and audit table independent, making it more scalable in terms of system size and (iii) although higher latency is expected due to its distributed deployment, around 70 per cent of user transactions will incur within acceptable latency intervals. The rest of the article is outlined as follows. In Section II, research background and related work is provided, followed by an explanation of both system and attack models in Section III. Next, in Section IV, we explain the architecture's formal specification, having role segregation, provenance, timelining and causality as $\mathcal{C}o\mathcal{C}$ properties [6]. Then, in sections V and VI, we deploy and evaluate the architecture performance. Finally, conclusions about our current and future work are given in Section VII.

II. Background and Related Work

The malicious insider threat against information [10][11][12] stored in databases urges the development of alternative solutions with auditing capabilities within distributed architectures even in the presence of an insider adversary. Although, such architectures have been already proposed [1][13][14][15][16], none of them considered Chain of Custody ($\mathcal{C}o\mathcal{C}$) properties and their implication in the admissibility of audit records as digital evidence. Our current research is focused on the design and

deployment of such an architecture, based on previous work developed in [6], where a vector-clock ($\mathbb{VC}$) mechanism[17] was used for tracking $\mathcal{DML}$ operations' provenance and causality, producing audit records within a forensically-ready database architecture. Nonetheless, beyond the evident scalability issue between the $\mathbb{VC}$ timestamp size and the number of audit tables, this mechanism also introduced precision issues and uncertain causal observations [18] because these are designed to enable operation ordering rather than, in fact, determining the actual physical time of their occurrence. Conversely, recent work in the field of multi-version databases such as Google's Spanner [19], Cockroach DB [9] and Yesquel [20] has inspired our current work with the introduction of a Hybrid-Logical-Clock ($\mathbb{HLC}$) mechanism which extends the functionality of vector clocks by using synchronised system clocks for calculating accurate timestamps of audit records. As a result, both the scalability and uncertainty problem of $\mathbb{VC}$ can be solved since our $\mathbb{HLC}$-based proposal employs more accurate timestamp calculations to build a timeline for analysing the causality of audit records during their generation, collection and preservation.

III. Models

We now present the models assumed in this work.

A. System Model

Our proposed architecture comprises a transactional ($\mathcal{N}_{\mathcal{DB}}$) and a forensic ($\mathcal{F}_{\mathcal{DB}}$) database for the *generation, collection and preservation* of admissible audit records. We assume $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$ to be sets of database tables and audit tables, respectively, being each $\mathcal{N}_{\mathcal{DB}}$ horizontally fragmented so that the architecture can scale to include more than one node (a pair $\langle \mathcal{N}_{\mathcal{DB}}, \mathcal{F}_{\mathcal{DB}} \rangle$) if required. Each database table $T_i \in \mathcal{N}_{\mathcal{DB}}$ has a corresponding audit table $\mathcal{F}_i \in \mathcal{F}_{\mathcal{DB}}$. The execution of the j^{th} $\mathcal{DML}$ operation (insert, update or delete) in a table $T_i \in \mathcal{N}_{\mathcal{DB}}$ initiates the generation, collection and preservation of an audit record e_i^j in the corresponding audit table $\mathcal{F}_i \in \mathcal{F}_{\mathcal{DB}}$. For maintaining consistency and atomicity in databases, not only for audit records, but also the resulting timeline, conservative 2-phase locking [21] is implemented as a single transactional block from the generation of $\mathcal{DML}$ operations at $\mathcal{N}_{\mathcal{DB}}$ to the collection and preservation of audit records at $\mathcal{F}_{\mathcal{DB}}$. We also assume that each node have *forensic controllers* for the accurate timestamping and sequencing of the records.

B. Attack Model

An *insider adversary model* is considered [10][22], assuming insiders misusing their access credentials to $\mathcal{N}_{\mathcal{DB}}$. Insiders interact with $\mathcal{N}_{\mathcal{DB}}$ using low-privilege ($\mathcal{DB}_{usr}$), or high-privilege ($\mathcal{DB}_{adm}$) roles. The first is for data entry purposes only whilst the second is a high-privilege role for managing $\mathcal{N}_{\mathcal{DB}}$ and its stored data. We assume that an insider can use the assigned $\mathcal{DB}_{adm}$ role to access $\mathcal{N}_{\mathcal{DB}}$, and execute $\mathcal{DML}$ operations to modify or impersonate other insiders with low privilege access ($\mathcal{DB}_{usr}$). Insiders can execute $\mathcal{DML}$ operations against any database table in $\mathcal{N}_{\mathcal{DB}}$, the actions of which can be monitored by a forensic role ($\mathcal{DB}_{for}$) exclusively reserved for accessing audit records and their timeline in $\mathcal{F}_{\mathcal{DB}}$. As insiders are not trusted, we require that $\mathcal{DB}_{adm} \neq \mathcal{DB}_{for}$ to prevent insiders from conveniently disabling forensic functionality. We also assume no collusion between a $\mathcal{DB}_{adm}$ and a $\mathcal{DB}_{for}$.

IV. Formalising the Proposed Architecture

To detect malicious insider actions within our proposed architecture, Chain-of-Custody ($\mathcal{CoC}$) must be enforced, requiring the following important properties as defined in [6]: (i) role segregation, (ii) $\mathcal{DML}$ operation provenance, (iii) event timelining and (iv) causality. As a result, audit records can be used as digital evidence to attribute malicious insider actions against a transactional database $\mathcal{N}_{\mathcal{DB}}$.

A. Role Segregation:

In the next definitions, a clear separation of duties is established, abstracting the forensic functionality to $\mathcal{N}_{\mathcal{DB}}$ users whilst preventing $\mathcal{F}_{\mathcal{DB}}$ users to interfere with the normal operations of $\mathcal{N}_{\mathcal{DB}}$.

Definition IV.1. *Transactional Database: A transactional database $\mathcal{N}_{\mathcal{DB}}$ is a set of tables $T_1 \dots T_m$:*

$$\mathcal{N}_{\mathcal{DB}} = \{T_i \mid 1 \leq i \leq m\} \tag{1}$$

Definition IV.2. *Forensic Database: A forensic database $\mathcal{F}_{\mathcal{DB}}$ is a set of audit tables $\mathcal{F}_1 \dots \mathcal{F}_m$:*

$$\mathcal{F}_{\mathcal{DB}} = \{\mathcal{F}_i \mid 1 \leq i \leq m\} \tag{2}$$

Definition IV.3. *Database Table: A database table T_i is a sequence of entries d_i^k, representing the k^{th} entry in table T_i:*

$$T_i = \left[d_i^k \mid 1 \leq k\right] \tag{3}$$

Definition IV.4. *Audit Table: An audit table is a sequence of audit records e_i^j, representing the j^{th} $\mathcal{DML}$ operation in table T_i:*

$$\mathcal{F}_i = \left[e_i^j \mid 1 \leq j\right] \tag{4}$$

Then, a function corr maps the current state of a database table $T_i \in \mathcal{N}_{\mathcal{DB}}$ with their corresponding audit table $\mathcal{F}_i \in \mathcal{F}_{\mathcal{DB}}$:

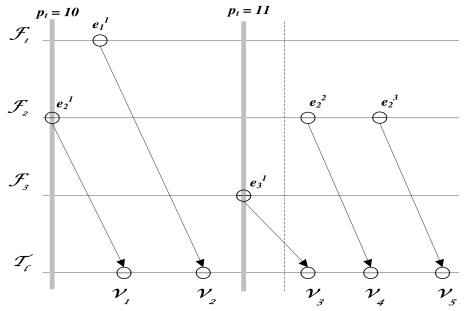
$$\begin{gathered} corr : \mathcal{N}_{\mathcal{DB}} \rightarrow \mathcal{F}_{\mathcal{DB}} \\ corr(T_i') = \mathcal{F}_i \end{gathered} \tag{5}$$

Definition IV.5. *Database Roles: Let $\mathcal{R}$ be a 3-element set denoting database roles such that:*

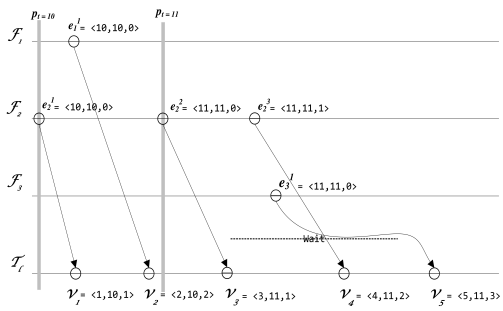
$$\mathcal{R} = \{\mathcal{DB}_{usr}, \mathcal{DB}_{adm}, \mathcal{DB}_{for}\} \tag{6}$$

Both roles $\mathcal{DB}_{usr}$ and $\mathcal{DB}_{adm}$ enable insiders to interact with $\mathcal{N}_{\mathcal{DB}}$ to perform regular operations (data entry) or administrative functions (data management), respectively. Meanwhile, $\mathcal{DB}_{for}$ is a role exclusively assigned to forensic users who can access $\mathcal{F}_{\mathcal{DB}}$ to monitor suspicious insider actions if required.

Definition IV.6. *Insider: Let $\mathcal{I}$ be a set of insiders who can interact with $\mathcal{N}_{\mathcal{DB}}$, and perform $\mathcal{DML}$ operations in a table $T_i \in \mathcal{N}_{\mathcal{DB}}$. Then, a surjective function $assgTo$ maps an insider*



(a) Audit records e_i^j in a $\mathbb{VC}$ timeline $T\iota$. At time $p_t = 11$ an inconsistent (dashed) cut could also be obtained, wrongfully suggesting that e_3^1 could have happened at $p_t = 10$.



(b) Audit records e_i^j in a $\mathbb{HLC}$ timeline $T\iota$. This mechanism solves $\mathbb{VC}$ inconsistency using a concurrency flag. E.g. 2 and 3 audit records have been certainly observed at $p_t = 10$ and $p_t = 11$, respectively

Fig. 1: Construction of a timeline $T\iota$ using Vector-Clocks ($\mathbb{VC}$) and Hybrid-Logical-Clocks ($\mathbb{HLC}$)

$\mathfrak{u}$ *to only one database role* $\mathfrak{r}$*, preventing, for example, having forensic and administrative roles assigned to the same insider.*

$$\begin{aligned} &assgTo : \mathcal{I} \twoheadrightarrow \mathcal{R} \\ &\forall \mathfrak{u} \in \mathcal{I} \bullet \exists \mathfrak{r} \in \mathcal{R} \bullet \mathfrak{r} = assgTo(\mathfrak{u}) \end{aligned} \tag{7}$$

B. $\mathcal{DML}$ Operation Provenance

For accountability purposes, the following definitions allow capturing provenance information about interactions between insiders $\mathfrak{u}$ and $\mathcal{N}_{\mathcal{DB}}$.

Definition IV.7. *$\mathcal{DML}$ Operations: Let $\mathcal{O}$ be a 3-element set of $\mathcal{DML}$ operations such that:*

$$\mathcal{O} = \{I, D, U\}, \textit{insert (I), delete (D), update (U)} \tag{8}$$

Definition IV.8. *Provenance: Let provenance $\mathcal{P}$ be a set of 5-attribute tuples ρ_i^k, capturing the UTC timestamp T_s, insider $\mathfrak{u}$, role $\mathfrak{r}$ and origin $\mathfrak{o}$ of a $\mathcal{DML}$ operation o_p in the k^{th} entry of table T_i:*

$$\begin{aligned} &\mathcal{P} = \{\rho_i^k \mid 1 \leq m_i \leq i, 1 \leq n_i \leq k\} \\ &\rho_i^k = \langle T_s, o_p, \mathfrak{u}, \mathfrak{r}, \mathfrak{o}\rangle, \textit{ where } T_s \in \mathbb{Z}, o_p \in \mathcal{O}, \mathfrak{u} \in \mathcal{I}, \mathfrak{r} \in \mathcal{R}, \\ &\qquad \mathfrak{o} \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \end{aligned} \tag{9}$$

Here, provenance is tuple-size independent, allowing for more/less contextual information if required.

C. Audit Record Timelining

A timeline $T\iota$ is an accurate chronological record for explaining the occurrence of $\mathcal{DML}$ operations in a table $T_i \in \mathcal{N}_{\mathcal{DB}}$, and preserving the occurrence of their corresponding audit records, generated and collected in an audit table $\mathcal{F}_i \in \mathcal{F}_{\mathcal{DB}}$. Our current proposal uses a Hybrid-Logical-Clock ($\mathbb{HLC}$) mechanism, an approach similar to the Vector-Clock ($\mathbb{VC}$) counterpart proposed in [6]. Whilst both approaches timeline and analyse the causality of audit records during their generation, collection and preservation, $\mathbb{VC}$ is not very efficient in distributed architectures due to both the lack of scalability and uncertain concurrent observations[9]. Firstly, $\mathbb{VC}$ cannot support an increasing number of audit tables without augmenting it timestamp size. Meanwhile, $\mathbb{HLC}$ is independent of the number of audit tables, making it more scalable to support more nodes $\langle \mathcal{N}_{\mathcal{DB}}, \mathcal{F}_{\mathcal{DB}} \rangle$ if required which makes it ideal for distributed fragmented or replicated databases. Secondly, although $\mathbb{VC}$ timestamps are efficient for tracking causality, they are not aware of physical time. Thus, as shown in Fig.1a, $\mathbb{VC}$ may not identify consistent cuts at physical time p_t, resulting in inaccurate observations of concurrent audit records. Whereas in Fig.1b, $\mathbb{HLC}$ solves this problem with a concurrency flag to monitor concurrent events on every observable time interval. In the following definitions, we explain the functional requirements to build a $\mathbb{HLC}$-based timeline $T\iota$, assuming both $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$ synchronised using a reliable time protocol such as NTP (Network Time Protocol) for accurate physical time reading.

Definition IV.9. *Audit Record: When a $\mathcal{DML}$ operation O in a table $T_i \in \mathcal{N}_{\mathcal{DB}}$ occurs, an audit record is created. Hence, an audit record e_i^j is a 3-tuple that captures the table state change from T_i to T_i' with timestamp τ:*

$$\begin{aligned} &e_i^j = \langle \tau, \delta, \rho_i^k \rangle, \textit{ where } \tau \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}, \\ &\qquad \delta : \textit{ tuple } \langle d_i^k, d_i^k\prime \rangle, \rho_i^k \in \mathcal{P} \end{aligned} \tag{10}$$

Notice that $e.\tau \neq e.\rho[T_s]$ in Eq. (10). Whilst the former captures the UNIX Epoch time as part of a $\mathbb{HLC}$ timestamp when changing table state, the latter captures the UTC timestamp T_s as a provenance attribute of e when a $\mathcal{DML}$ operation $e.\rho[\mathfrak{o}]$ causes that state shift. Further details about the calculation of $e.\tau$ and $e.\rho[T_s]$ are explained in Sections V-C2 and V-C3.

Definition IV.10. *Evidence Generation: A function gen maps a k^{th} entry on table T_i with the corresponding $\mathcal{DML}$ operation that changed the state of such table to T_i':*

$$\begin{aligned} &gen : \mathbb{N}^+ \times \mathcal{N}_{\mathcal{DB}} \rightarrow \mathcal{O} \times \mathcal{N}_{\mathcal{DB}} \\ &gen(k, T_i) = \begin{cases} a)\ (I, T_i'), & \textit{if } T_i' = T_i \frown \left[d_i^k\right] \\ b)\ (U, T_i'), & \textit{if } T_i' = T_i \oplus \left[d_i^k\prime\right] \\ c)\ (D, T_i'), & \textit{if } T_i' = \{k\} \lhd T_i \end{cases} \end{aligned} \tag{11}$$

a) During an insertion, an entry d is appended to a table T.

b) In case of an update, an entry d in table T is overwritten.
c) If an entry d is deleted, it is removed from table T.

Definition IV.11. *Evidence Collection: Following Eq.(5) and Eq. (11), in Eq.(12), a function colct maps a $\mathcal{DML}$ operation on the k^{th} entry of table T_i with the j^{th} audit record in its corresponding audit table $\mathcal{F}_i$, recording the table change from T_i to T_i':*

$$colct : \mathcal{O} \times \mathcal{N}_{\mathcal{DB}} \rightarrow \mathcal{O} \times \mathcal{F}_{\mathcal{DB}}$$

$$colct(gen(k, T_i)) = \begin{cases} a)\ (I, corr(T_i')), \text{if } \mathcal{F}_i' = \mathcal{F}_i \frown [e_i^j] \wedge \\ e_i^j = \langle \tau, \perp, d_i^k, \rho_i^k \rangle \\ b)\ (U, corr(T_i')), \text{if } \mathcal{F}_i' = \mathcal{F}_i \frown [e_i^j] \wedge \\ e_i^j = \langle \tau, d_i^k, d_i^k\prime, \rho_i^k \rangle \\ c)\ (D, corr(T_i')), \text{if } \mathcal{F}_i' = \mathcal{F}_i \frown [e_i^j] \wedge \\ e_i^j = \langle \tau, d_i^k, \perp, \rho_i^k \rangle \end{cases} \tag{12}$$

a) If an entry d is inserted, an audit record e is appended to an audit table $\mathcal{F}$, and $e.\delta = \langle \perp, d_{new} \rangle$.
b) If there is an update, an audit record e is appended to an audit table $\mathcal{F}$, and $e.\delta = \langle d_{old}, d_{new} \rangle$.
c) In case an entry d is deleted, an audit record e is appended to an audit table $\mathcal{F}$, and $e.\delta = \langle d_{old}, \perp \rangle$.

Definition IV.12. *Evidence Preservation: A causal table $T\iota \in \mathcal{F}_{\mathcal{DB}}$ is a timeline comprising a sequence of timestamps $\mathcal{V}_m$, such that:*

$$T\iota = [\mathcal{V}_m \mid 1 \leq m \leq n] \tag{13}$$

A timestamp $\mathcal{V}_m$ is a 4-tuple, such that:

$$\mathcal{V}_m = \langle i, j, \tau^m, T_s^m \rangle, \text{ where } \tau^m \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}, T_s^m \in \mathbb{Z} \tag{14}$$

A timestamp $\mathcal{V}_m$ captures the generation and collection of the j^{th} audit record in the i^{th} audit table with $\mathbb{HLC}$ timestamp τ^m and UTC timestamp T_s^m.

Notice that $e.\tau$ in Eq.(12) $<$ $e.\tau^m$ in Eq.(14) so that the observed timestamp τ^m of the audit record e_i^j in $T\iota$ is always greater than the registered timestamp τ in its audit table $\mathcal{F}_i$.

Lastly, from Eq. (11) and Eq. (12), in Eq.(15) a function $prsv$ appends the timestamp $\mathcal{V}_m$ to the timeline $T\iota$ if an entry d is in the tuple $e.\delta$ as either a d_{old} or d_{new} entry.

$$\begin{aligned} prsv &: \mathcal{O} \times \mathcal{F}_{\mathcal{DB}} \rightarrow \mathcal{F}_{\mathcal{DB}} \\ prsv(colct(gen(k, T_i))) &= T\iota \frown [\langle i, j, \tau^l, T_s^l \rangle] \\ &\iff \exists! \delta \in e_i^j.\delta \bullet \delta \neq \langle \emptyset \rangle \end{aligned} \tag{15}$$

Notice that by means of composite functions $colct$ and $prsv$ we are enforcing conservative 2-phase locking during the generation, collection and preservation of audit records. This simply means that audit records cannot be preserved if their generation and collection are not executed in that order.

D. Event Causality

As mentioned before, using $\mathbb{HLC}$ to build a timeline offers evident advantages for designing an efficient distributed forensically-aware database architecture. We develop our current work based on the causality definitions and $\mathbb{HLC}$ implementation given in [9]. Unlike the $\mathbb{VC}$ causality definitions in [6], their counterparts featured in our current work strengthen their $\mathbb{HLC}$ timestamp conditions because these cannot increase unboundedly, ending up recording audit records ahead of the timeline's physical time. Now, once audit records have been timelined, the index j used to trace their ordering within an audit table $\mathcal{F}_i$ can be omitted so that audit records can be referred as e_i being identifiable only by their $\mathbb{HLC}$ timestamp and the index of the audit table $\mathcal{F}_i$ to which they belong.

Definition IV.13. *Forensic Evidence Set: Audit records in $T\iota \in \mathcal{F}_{\mathcal{DB}}$ are elements of an forensic evidence set $\mathcal{E}$:*

$$\mathcal{E} = \{e_i \mid i \in \mathbb{N}^+\} \tag{16}$$

Causality in these audit records dictate that they are either sequential or concurrent, justifying their relationship by means of a "happen-before" relation ($\xrightarrow{hb}$). So when using hybrid logical clocks ($\mathbb{HLC}$), from Eq. (10), $e_i.\tau$ is 3-tuple recording the following time information [9]:

- *$e_i.\tau[p_t]$: the instant physical time of a $\mathcal{DML}$ operation observed and recorded in an audit record e_i.*
- *$e_i.\tau[l]$: the maximum physical time known about the occurrence of a $\mathcal{DML}$ operation in the corresponding audit record e_i.*
- *$e_i.\tau[c]$: a flag to indicate that an audit record e_i has recorded a $\mathcal{DML}$ operation which is concurrent with another.*

Definition IV.14. *Event Sequentiality Property: Let $e_a, e_b \in \mathcal{E}$ be two audit records collected in audit tables $\mathcal{F}_a$ and $\mathcal{F}_b$, recording the $\mathcal{DML}$ operations generated in their corresponding database tables T_a and T_b. Then, e_a is sequential with e_b if $(e_a.\tau[l], e_a.\tau[c]) < (e_b.\tau[l], e_b.\tau[c])$:*

$$\begin{aligned} \forall e_a, e_b \in \mathcal{E} \bullet (e_a \xrightarrow{hb} e_b \Rightarrow (e_a.\tau[l], e_a.\tau[c]) < \\ (e_b.\tau[l], e_b.\tau[c]) \iff (e_a.\tau[l] < e_b.\tau[l]) \vee ((e_a.\tau[l] \\ = e_b.\tau[l]) \wedge (e_a.\tau[c] < e_b.\tau[c]))) \end{aligned} \tag{17}$$

Definition IV.15. *Event Transitivity Property: Having three audit records $e_a, e_b, e_c \in \mathcal{E}$, if $e_a \xrightarrow{hb} e_b$ and $e_b \xrightarrow{hb} e_c$; then, $e_a \xrightarrow{hb} e_c$:*

$$\begin{aligned} \forall e_a, e_b, e_c \in \mathcal{E} \bullet (e_a \xrightarrow{hb} e_b \wedge e_b \xrightarrow{hb} e_c \Rightarrow e_a \xrightarrow{hb} e_c \wedge \\ (e_a.\tau[l], e_a.\tau[c]) < (e_c.\tau[l], e_c.\tau[c])) \end{aligned} \tag{18}$$

Which follows from Eq.(17) for demonstrating that $e_a \xrightarrow{hb} e_c$.

Definition IV.16. *Event Concurrency Property: The concurrency property applies to all audit records $e_i \in \mathcal{E}$ recording*

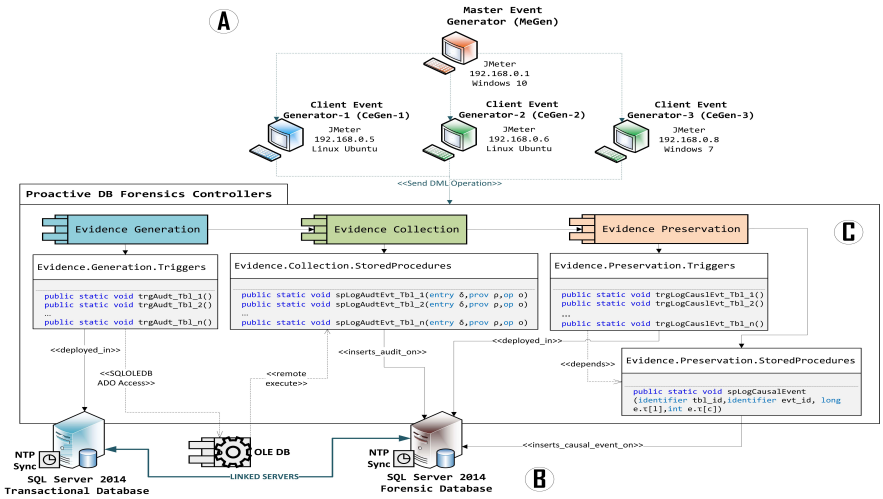


Fig. 2: Experimental Forensically-Aware Database Architecture.

$\mathcal{DML}$ operations generated in their corresponding tables $T_i \in \mathcal{N}_{\mathcal{DB}}$, which are not sequential with each other:

$$\forall e_a, e_b \in \mathcal{E} \bullet (e_a \parallel e_b \Rightarrow \neg(e_a \xrightarrow{hb} e_b \wedge e_b \xrightarrow{hb} e_a)) \wedge (e_a.\tau[c], e_b.\tau[c] > 0) \quad (19)$$

V. Experimental Deployment

For audit records $e_i^j \in \mathcal{F}_i$ to be used as digital evidence, Chain-of-Custody ($\mathcal{CoC}$) must be initiated and maintained during their generation, collection and preservation within a forensically-aware database architecture. In Fig.2, we implement an experimental architecture, based on the $\mathbb{VC}$ approach proposed in [6], yet utilizing a $\mathbb{HLC}$ mechanism instead, similar to the one implemented by [9]. As a result, a timeline $T\iota$ of timestamps $\mathcal{V}_n$ can be built to keep an audit trail of the occurrence of $\mathcal{DML}$ operations against $\mathcal{N}_{\mathcal{DB}}$. The functional components of the experimental architecture are described as follows:

(A) Concurrent $\mathcal{DML}$ Operation Generator: *JMeter* configured in Master-Slave mode.
(B) Transactional ($\mathcal{N}_{\mathcal{DB}}$) and Forensic ($\mathcal{F}_{\mathcal{DB}}$) Databases: Deployed in *MSSQL Server 2014* in Linked Server mode and synchronised with an NTP-based time service.
(C) Proactive Database Forensics Controllers: *SQL C# triggers and stored procedures* implemented as forensic controllers in both $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$.

For better understanding, it is important to clarify the difference between *architectural components* and *functional controllers* in our research context. Whilst *a component* refers to an architectural element in charge of performing specific actions, *a controller* is a logical routine responsible for coordinating the behaviour of such components during the generation, collection and preservation of audit records. In the following subsections, the architecture's functional components, including the logic functionality of its forensic controllers, are explained in detail.

A. Concurrent $\mathcal{DML}$ Operation Generator

For simulating real user transactions, synthetic workload is generated by implementing a concurrent environment in which a set of n insiders $\in \mathcal{I}$ (Def.IV.6) can perform a number of *DML operations* against $\mathcal{N}_{\mathcal{DB}}$ using a specific administrative role ($\mathcal{DB}_{adm}$ or $\mathcal{DB}_{us}$) (Def.IV.5). Thus, for stressing the architecture, a *Master Event Generator* (MeGen) and three *Client Event Generators* (CeGen) are used, as shown in Fig.2.[sec. (A)]. These generators are deployed using *JMeter* in master-slave mode, enabling us to open threaded database connections for sending concurrent $\mathcal{DML}$ *operations* to $\mathcal{N}_{\mathcal{DB}}$.

B. Transactional and Forensic Databases

For preventing audit records to be tampered with due to potential malicious insider actions, *role segregation* (Section IV-A) is required so that audit and forensic activities can be performed in a transparent, yet efficient way. Unlike the centralised $\mathbb{VC}$-based architecture in [6], ours achieve this by distributing and *physically segregating transactional and forensic operations* using two different database servers, each hosting $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$ with explicit operative ($\mathcal{DB}_{adm}, \mathcal{DB}_{us}$) and forensic ($\mathcal{DB}_{for}$) roles, respectively. For facilitating both the collection of audit records in each audit table $\mathcal{F}_m$, and the construction of the timeline $T\iota$, $\mathcal{F}_{\mathcal{DB}}$ (Fig.3) is de-normalised. This ensures scalability in more complex environments where centralised architectures may not perform well [9].

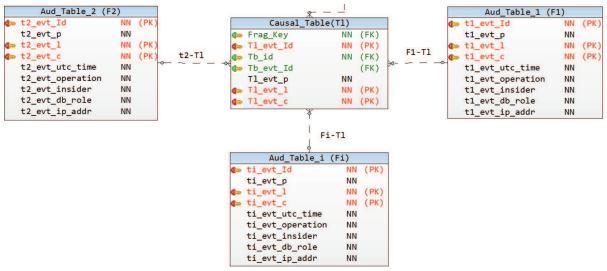


Fig. 3: A de-normalised forensic database structure.

C. Proactive Database Forensic Controllers

SQL CLR C# triggers and stored procedures are implemented as forensic controllers, and deployed in both $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$, beginning their execution when *the Concurrent $\mathcal{DML}$ Operation Generator* component sends threaded operations to $\mathcal{N}_{\mathcal{DB}}$. These controllers, as depicted in Fig.2.[sec. (C)] are implemented following the formal specifications given in Section IV, allowing $\mathcal{CoC}$ to be initiated and maintained since the generation and collection of audit records until their preservation in the timeline $T\iota$. As opposed to the $\mathbb{VC}$ architecture in [6], the $\mathbb{HLC}$ mechanism in these controllers, makes our architecture more scalable because the timestamp size in $T\iota$ is independent of the number of audit tables. Nonetheless, as previously shown in Fig.1b, due to transitivity, there is a high possibility of having intermediate audit records (Eq.18), observed in $T\iota$ at the same p_t time. This concurrency problem may lead to inconsistencies requiring controlled access to $T\iota$ by serialising the generation, collection and preservation of audit records [6]. Therefore, any intermediate audit record requiring to be timelined must 'wait' until any previously observed ones are recorded in $T\iota$. Lastly, due to $\mathbb{HLC}$ implementation requirements, these databases are synchronised with an *NTP-based time service* every 15 min., and deployed as *linked (or federated) database servers* for enabling connectivity between $\mathcal{N}_{\mathcal{DB}}$ and $\mathcal{F}_{\mathcal{DB}}$, and for remotely executing these controllers. In the following sections, the generation, collection and preservation of audit records through these controllers are explained in depth.

Algorithm 1 Evidence Generation Controller

vars:
$T_s : UTC_{now}$ ▷ DateTime in UTC format
$o_p : \mathcal{O}$ ▷ $\mathcal{DML}$ operation
$\mathfrak{u} : \mathcal{I}$ ▷ insider
$\mathfrak{o} : ip\_loc$ ▷ Originating IP address
$\rho_i^k : \mathcal{P}$ init $\langle\rangle$ ▷ Provenance in Eq.9
$\delta :$ tuple $\langle d_{old}, d_{new}\rangle$ init $\langle\rangle$ ▷ T_i transition
actions:
while $\exists\ gen(k, T_i)$ **do** ▷ Eq.11
  if (I, T_i') **then** ▷ Insert
    $\delta := \langle \perp, d_i^k \rangle$
    $o := I$
  end if
  if (U, T_i') **then** ▷ Update
    $\delta := \langle d_i^k, d_i^k\prime \rangle$
    $o := U$
  else ▷ Delete
    $\delta := \langle dk_i^k, \perp \rangle$
    $o := D$
  end if
  $\rho_i^k := \langle T_s, o_p, \mathfrak{u}, assgTo(\mathfrak{u}), \mathfrak{o}\rangle$ ▷ Def.IV.5 role
  $send(\delta, \rho_i^k, colct(gen(k, T_i)))$ ▷ call to colct
end while

1) Evidence Generation Controller: This controller is comprised of *evidence generation triggers* deployed in $\mathcal{N}_{\mathcal{DB}}$ to be executed every time the *Concurrent $\mathcal{DML}$ Operation Generator* sends an operation to $\mathcal{N}_{\mathcal{DB}}$. As shown in Alg.1, this controller captures and remotely sends to its corresponding audit table, provenance information (Def. IV.8) about the $\mathcal{DML}$ Operation that either created or affected an entry d_i^k in a table T_i (Def. IV.10).

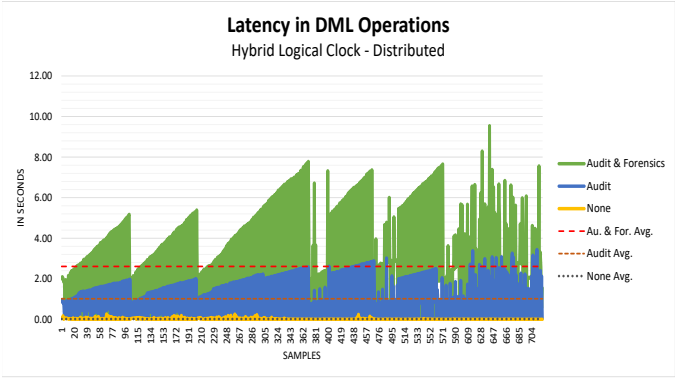
2) Evidence Collection Controller: After the *evidence generation controller* has been executed, *evidence collection stored procedures* deployed in $\mathcal{F}_{\mathcal{DB}}$ are in charge of both receiving provenance information about $\mathcal{DML}$ operations on a table T_i, and storing audit records in its corresponding audit table $\mathcal{F}_i$. Besides the received provenance information, the

Algorithm 2 Evidence Collection Controller

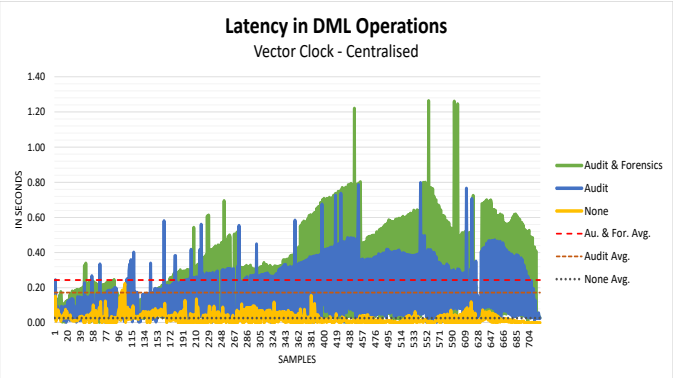
vars:
$\tau : \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}$ init $\perp$ ▷ $\mathbb{HLC}$ timestamp
$p_t : \mathbb{Z}$ init 0 ▷ Physical clock time
$l, l' : \mathbb{Z}$ init 0 ▷ Max. physical time p_t known so far.
$c : \mathbb{N}$ init 0 ▷ Concurrency tracking flag.
$j : \mathbb{Z}$ init 1 ▷ Audit record local order
$e_i^j :$ tuple $\langle \tau, \delta, p\rangle$ init $\langle\rangle$ ▷ Eq.10
actions:
if $recv(\delta, \langle T_s, o_p, \mathfrak{u}, \mathfrak{r}, \mathfrak{o}\rangle, o_p, \mathcal{F}_i)$ **then**
  $p_t := toUNIXtime(T_s)$
  $l' := l$
  $l := max(l', p_t)$
  if $l = l'$ **then** ▷ if there are concurrent $\mathcal{DML}$ ops.
    $c := c + 1$
  else
    $c := 0$
  end if
  $\tau := \langle p_t, l, c\rangle$ ▷ $\mathcal{DML}$ Operation $\mathbb{HLC}$ timestamp
  $T_s := toUTCtime(l + c)$ ▷ $\mathcal{DML}$ operation UTC Timestamp
  $e_i[j] := \langle \tau, \delta, \langle T_s, o_p, \mathfrak{u}, \mathfrak{r}, \mathfrak{o}\rangle\rangle$
  $\mathcal{F}_i' := \mathcal{F}_i \frown [e_i^j]$
  $j := size(\mathcal{F}_i) + 1$
end if
$send(i, j, e.\tau[l], e.\tau[c], prsv(o_p, \mathcal{F}_i'), T\iota)$ ▷ call to prsv

$\mathbb{HLC}$ mechanism implemented in Alg.2, enables this controller to capture a timestamp τ when a table changes state from T_i to T_i'. As a result, an accurate UTC timestamp T_s about the occurrence of $\mathcal{DML}$ operations can also be stored in audit records, using for its calculation the value of τ captured by the controller.

3) Evidence Preservation Controller: This controller is composed of *evidence preservation triggers* and a particular *stored procedure* which also implements a $\mathbb{HLC}$-based *forensic mechanism* for building the timeline $T\iota$. These triggers execute the stored procedure every time the controller is notified of $\mathcal{DML}$ operations being generated in a table $T_i \in \mathcal{N}_{\mathcal{DB}}$, and collected in its corresponding audit table $\mathcal{F}_i \in \mathcal{F}_{\mathcal{DB}}$. Therefore, following Alg.3, the resulting timeline $T\iota$ can be used later as an audit trail for explaining the causality of audit records, and the attribution of $\mathcal{DML}$ operations performed against $\mathcal{N}_{\mathcal{DB}}$.



(a) Latency in the Distributed $\mathbb{HLC}$-based DB forensics Architecture.



(b) Latency in the Centralised $\mathbb{VC}$-based DB forensics Architecture.

Fig. 4: Latency Results - Vector-Clocks ($\mathbb{VC}$) vs. Hybrid-Logical-Clocks ($\mathbb{HLC}$)

Algorithm 3 Evidence Preservation Controller

vars:
$\tau^n : \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}$ init $\perp$ ▷ $\mathbb{HLC}$ timestamp
$T_s^n : \mathbb{Z}$ init 0
$p_t : \mathbb{Z}$ init 0 ▷ Physical clock time
$l, l', l_m : \mathbb{Z}$ init 0 ▷ Max. physical time p_i known so far.
$c, c_m : \mathbb{N}$ init 0 ▷ Concurrency tracking flag.
$n : \mathbb{Z}$ init 1 ▷ Timestamp counter
$\mathcal{V}_n$: tuple $\langle i, j, \tau^n, T_s^n \rangle$ ▷ Eq.14 - Audit record timestamp in $T\iota$
if $recv(i, j, e.\tau[l], e.\tau[c], prsv(o_p, \mathcal{F}_i'), T\iota)$ **then**
  $p_t := UNIXtime_{now}$
  $l\prime := l$
  $l_m := e.\tau[l]$
  $c_m := e.\tau[c]$
  $l := max(l\prime, l_m, p_t)$
  if $l = l' = l_m$ **then**
    $c := max(c, c_m) + 1$
  else
    if $l = l'$ **then**
      $c := c + 1$
    else
      if $l = l_m$ **then**
        $c := c_m + 1$
      else
        $c := 0$
      end if
    end if
  end if
  $\tau^n := \langle p_t, l, c \rangle$ ▷ $\mathbb{HLC}$ timestamp
  $T_s^n := toUTCtime(l + c)$ ▷ UTC timestamp
  $\mathcal{V}[n] := \langle i, j, \tau^n, T_s^n \rangle$ ▷ Audit record timestamp in $T\iota$
  $T\iota' := T\iota \frown [\mathcal{V}_n]$
  $n := size(T\iota) + 1$
end if

VI. Evaluation

For evaluation purposes, three stress test scenarios were considered:

- With *audit & forensic controls*: Evidence Generation and Preservation Triggers are activated for generating, collecting and preserving the occurrence of audit records.
- With *audit controls only*: Evidence Generation Triggers enabled to generate and collect audit records.
- With *no controls* enabled: There is no audit or forensic controllers activated.

During the test, 720 samples of $\mathcal{DML}$ operations (per test scenario) were simulated using *JMeter* (Section V-A). The generation, collection and preservation controllers (Section V-C) were executed per each sample as serialized transactions (2-phase locking execution) to avoid uncontrolled concurrent observations.

TABLE I: Acceptable Latency Interval for Audit & Forensic Controls

Latency Interval (sec.)	No. of Samples	%
Between 0.021 and 2.610	408	56.7
Between 2.610 and 3.0	54	7.5
Between 3.0 and 3.50	30	4.2
Between 3.50 and 9.545	228	31.7
Total	*720*	*100.0*

As shown in Fig.4, increased latency was expected in our $\mathbb{HLC}$-based architecture when compared with its $\mathbb{VC}$ counterpart; mainly for two reasons (a) because it works in a *distributed environment* with two different database servers for transactional and forensic purposes each, and (b) due to inherent limitations when using linked (federated) communication that was mainly designed for data source merging between two servers[23], and not for heavy transactional workload such as the one implemented in the controllers. Nonetheless, assuming an acceptable latency interval (up to 3.5 sec. per transaction), in Table I, almost 70 per cent of samples showed latency between 0.021 and 3.5 sec. when *audit and forensic controls* were enabled. This proves that our $\mathbb{HLC}$-based architecture is

TABLE II: Provenance and Causality of Audit Records in $\mathcal{F}_{\mathcal{DB}}$

n	i	j	p_t	l	c	l_m	c_m	o_p	*Insider* $\mathfrak{u}$	*Role* $\mathfrak{r}$	*Origin* $\mathfrak{o}$	$T_s[n]_{UTC}$
161	2	31	1525135633234	1525135633234	0	1525135632969	0	I	1705997013	dbadmin	192.168.0.5	01:47:13.2341027 +01.00
162	2	32	1525135633234	1525135633234	1	1525135632969	0	I	1705997014	dbadmin	192.168.0.6	01:47:13.2341028 +01:00
163	6	16	1525135633234	1525135633234	1	1525135632986	0	I	1705997013	dbuser	192.168.0.5	01:47:13.2341029 +01:00
164	4	20	1525135633265	1525135633265	0	1525135633004	0	I	1705687116	dbuser	192.168.0.8	01:47:13.2653852 +01:00
165	2	33	1525135633294	1525135633294	0	1525135633003	0	I	1715987319	dbadmin	192.168.0.8	01:47:13.2944992 +01:00

Audit records $e_2^j \in \mathcal{F}_2$ Audit records $e_4^j \in \mathcal{F}_4$ Audit records $e_6^j \in \mathcal{F}_6$

resilient enough to work under heavy transactional workload in distributed settings.

VII. Conclusions

Both $\mathbb{VC}$ and $\mathbb{HLC}$-based proposals prevents audit record modification by capturing provenance of $\mathcal{DML}$ operations within a de-normalised segregated forensic database $\mathcal{F}_{\mathcal{DB}}$. Likewise, our architecture can also detect potential insider misuse by querying audit tables $\mathcal{F}_i$ and the timeline $T\iota$. For instance timestamps V_{161} and V_{163} (Table II), warns of an insider $\mathfrak{u}$ interacting with $\mathcal{N}_{\mathcal{DB}}$ from the same originating IP address, but using 2 different database roles. In contrast, as opposed to its $\mathbb{VC}$ centralised counterpart, our distributed $\mathbb{HLC}$ architecture has shown (i) higher accuracy when tracking $\mathcal{DML}$ operations' provenance and causality by timestamping them with accurate physical time readings for better concurrent event detection (Table II; V_{162}, V_{163}); (ii) more scalability in terms of system size when more than one node $\langle \mathcal{N}_{\mathcal{DB}}, \mathcal{F}_{\mathcal{DB}} \rangle$ is involved in the computation since timestamps $\mathcal{V}_m$ are independent of the number of audit tables; (iii) acceptable performance under heavy transactional workload, considering that 70 per cent of samples were processed with latency no higher than 3.5 sec. when audit and forensic capabilities were enabled. These results prove that $\mathbb{HLC}$ is very resilient to technical limitations, such as implementing conservative 2PL with linked servers. Future work will be developed towards formally proving the correcteness of our proposal.

Acknowledgements

This research was funded by the Secretariat of Higher Education, Science, Technology and Innovation of the Republic of Ecuador under Award No.: 043-CIBAE-2015.

References

[1] B. Bhargava, E. Mafla, and J. Riedl, "Push: An experimental facility for implementing distributed database services in operating systems," *Journal of Systems Integration*, vol. 3, no. 1, pp. 5–21, Mar 1993. [Online]. Available: https://doi.org/10.1007/BF01974169

[2] Microsoft, "Sql server audit (database engine)," *SQL Server 2017*, 2016, [Accessed 09 May 2018]. [Online]. Available: https://bit.ly/2FY7tsr

[3] Oracle, "Database auditing: Security considerations," *Database Security Guide*, 2018, [Accessed 09 May 2018]. [Online]. Available: https://bit.ly/2IovWfF

[4] W. Hauger and M. Olivier, "Forensic attribution in nosql databases," in *2017 Information Security for South Africa (ISSA)*, Aug 2017, pp. 74–82.

[5] M. S. Olivier, "On metadata context in database forensics," *Digital Investigation*, vol. 5, no. 3, pp. 115 – 123, 2009. [Online]. Available: https://bit.ly/2I91gvM

[6] D. A. Flores and A. Jhumka, "Implementing chain of custody requirements in database audit records for forensic purposes," in *2017 IEEE Trustcom/BigDataSE/ICESS*, Aug 2017, pp. 675–682.

[7] Q. Huang and L. Liu, "A logging scheme for database audit," *2009 Second International Workshop on Computer Science and Engineering*, vol. 2, pp. 390–393, 2009.

[8] Association of Chief Police Officers. (2012) Good Practice Guide for Digital Evidence. [Accessed 09 May 2018]. [Online]. Available: https://goo.gl/UUHFwQ

[9] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, "Logical physical clocks," in *Principles of Distributed Systems*. Cham: Springer International Publishing, 2014, pp. 17–32.

[10] G. Magklaras and S. Furnell, "Insider threat prediction tool: Evaluating the probability of it misuse," *Computers and Security*, vol. 21, no. 1, pp. 62 – 73, 2001. [Online]. Available: https://bit.ly/2K7RvOG

[11] G. Saathoff, T. Nold, and C. Holstege, "We have met the enemy and they are us: Insider threat and its challenge to national security," in *Strategic Intelligence Management*. Butterworth-Heinemann, 2013, pp. 24 – 35. [Online]. Available: https://bit.ly/2G2fhtm

[12] M. Walport, "Forensic science and beyond," *Government Office for Science*, 2015, [Accessed 09 May 2018]. [Online]. Available: https://bit.ly/2wlRcxQ

[13] B. Kogan and S. Jajodia, "An audit model for object-oriented databases," in *Proceedings Seventh Annual Computer Security Applications Conference*, Dec 1991, pp. 90–99.

[14] W. Lu and G. Miklau, "Auditing a database under retention restrictions," in *2009 IEEE 25th International Conference on Data Engineering*, March 2009, pp. 42–53.

[15] A. M. Rashad, "Detection and prevention of malicious activities on rdbms relational database management systems," 2012.

[16] B. U. S. Jadhav Shital, Kardile Bhagyashri, "Forensic investigation for database tampering using audit logs," 2015.

[17] O. Babaoğlu and K. Marzullo, "Distributed systems (2nd ed.)." ACM Press/Addison-Wesley Publishing Co., 1993, ch. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms, pp. 55–96. [Online]. Available: https://bit.ly/2rxveTN

[18] S. Yingchareonthawornchai, D. N. Nguyen, V. T. Valapil, S. S. Kulkarni, and M. Demirbas, "Precision, recall, and sensitivity of monitoring partially synchronous distributed systems," *CoRR*, vol. abs/1607.03369, 2016. [Online]. Available: http://arxiv.org/abs/1607.03369

[19] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, and C. F. et al., "Spanner: Google's globally-distributed database," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, 2012, pp. 261–264. [Online]. Available: https://bit.ly/2rqBpZR

[20] M. K. Aguilera, J. B. Leners, R. Kotla, and M. Walfish, "Yesquel: scalable SQL storage for web applications," *CoRR*, vol. abs/1411.2160, 2014. [Online]. Available: http://arxiv.org/abs/1411.2160

[21] A. Macdonald, "Deadlock in database systems," *Concurrency Control Mechanisms*, 2011, [Accessed 09 May 2018]. [Online]. Available: https://bit.ly/2FW2JUd

[22] R. C. D. Jr., A. J. Ferguson, and D. M. Cappelli, "Introduction to insider threat modeling, detection, and mitigation track," in *2012 45th Hawaii International Conference on System Sciences*, Jan 2012, pp. 2381–2381.

[23] Microsoft. (2017) Linked Servers (Database Engine). [Accessed 09 May 2018]. [Online]. Available: https://bit.ly/2HZLjf3